Università di Pisa
Dipartimento di Informatica

Technical Report

Modelling the behaviour of management operations in TOSCA

Antonio Brogi, Andrea Canciani, and Jacopo Soldani

Department of Computer Science, University of Pisa
{*surname*}*@di.unipi.it*

July 10, 2015

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Modelling the behaviour of management operations in TOSCA

Antonio Brogi, Andrea Canciani, and Jacopo Soldani

Department of Computer Science, University of Pisa
{*surname*}*@di.unipi.it*

Abstract

Managing complex applications over heterogeneous clouds is one of the emerging problems in the cloud era. The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) aims at solving this problem by providing a language to describe and manage complex cloud applications in a portable and vendor-agnostic way. TOSCA permits to define an application as an orchestration of components, whose types can specify states, requirements, capabilities and management operations — but not how they interact with each other.

In [1, 2] , we already discussed a simple extension of TOSCA that permits describing the behaviour of a component's management operations and their relations with its states, requirements, and capabilities. The objective of this short report is to show how to enrich the TOSCA modelling language to provide such extension.

1 Introduction

How to flexibly manage applications over heterogeneous clouds is an hot, open issue. In this perspective, OASIS released TOSCA (*Topology and Orchestration Specification for Cloud Applications* [4, 5]), a standard to support the automated management of complex cloud-based applications. TOSCA provides a modelling language to describe, in a portable and vendor-agnostic way, a cloud application and its management. An application is defined by instantiating component types, and by connecting a component's requirements to the capabilities of other components. Its management can then be described by orchestrating the operations of its components into workflow plans.

Unfortunately, the current version of TOSCA [4] does not permit to specify the behaviour of a cloud application's management operations. More precisely, it is not possible to describe the order in which the management operations of a component must be invoked, nor how those operations depend on the requirements or how they affect the capabilities of that component (and hence the requirements of other components they are connected to). This implies that the verification of whether a management plan is valid can only be performed manually, with a time-consuming and error-prone process.

We already discussed how to extend TOSCA so as to specify the management behaviour of TOSCA application components [1, 2]. Namely, the management protocols of a component can be described by means of a finite state machine

whose states and transitions are associated with conditions on the component's requirements and capabilities. Intuitively, the objective of those conditions is to define the consistency of a component's states and to constrain the executability of its operations to the satisfaction of its requirements.

The objective of this short report is to show how management protocols can be concretely represented in TOSCA. More precisely, after generalising our previous notion of management protocols [1, 2], we illustrate how to enrich the TOSCA modelling language to represent such protocols.

The rest of the report is organized as follows. Sect. 2 introduces TOSCA, and Sect. 3 discuss how TOSCA can be extended to model the behaviour of management operations. Sect. 4 draws some concluding remarks. Finally, the Appendix shows how the (current version of) TOSCA YAML Simple Profile can be extended to model the behaviour of management operations.

2 Background: TOSCA

TOSCA [4] is an emerging standard aimed at enabling the specification of portable cloud applications and the automation of their management. To do so, TOSCA provides a modelling language to describe the structure of a cloud application as a typed topology graph, and its tasks as plans. More precisely, each cloud application is represented as a `ServiceTemplate` (Fig. 1), consisting of a mandatory `TopologyTemplate` and of optional management `Plans`. Generic type definitions are also contained in the document defining the `ServiceTemplate` as they are referred to by the elements in its topology.

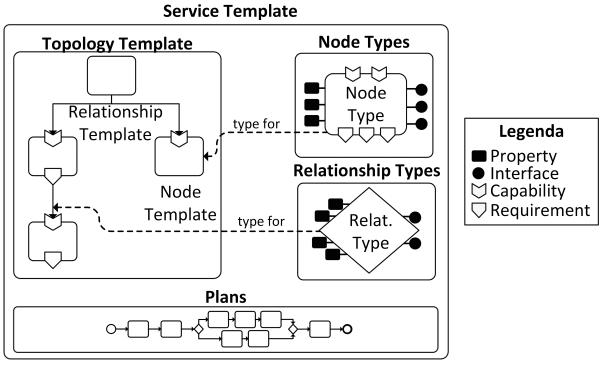

Fig. 1: TOSCA `ServiceTemplate`.

The `TopologyTemplate` is a typed directed graph describing the structure of the composite cloud application. Its nodes (`NodeTemplate`s) model the application components, while its edges (`RelationshipTemplate`s) model the relations among those components. `NodeTemplate`s and `RelationshipTemplate`s are typed by means of `NodeType`s and `RelationshipType`s, respectively. A `NodeType` defines (i) the observable properties of an application component, (ii) the possible states of its instances, (iii) its requirements, (iv) the capabilities it offers to satisfy other components' requirements, and (v) its management operations. `RelationshipType`s describe the properties of relationships occurring among components. Syntactically, properties are described by `PropertiesDe-`

finition, states by InstanceStates, requirements by RequirementDefinitions (of certain RequirementTypes), capabilities by CapabilityDefinitions (of certain CapabilityTypes), and operations by Interfaces and Operations.

Plans instead allow to describe the management aspects of a ServiceTemplate. More precisely, each Plan is a workflow orchestrating the management Operations offered by the application components to address (part of) the management of the whole cloud application[1].

3 Management protocols for cloud applications

Let N be a TOSCA NodeType, and S_N, R_N, C_N, and O_N be the finite sets of its states, requirements, capabilities, and management operations, respectively.

As we discussed in [1, 2], we want to describe whether and how the management operations of N depend on (i) other operations of the same node and/or on (ii) operations of other nodes providing the capabilities that satisfy the requirements of N.

The first kind of dependencies can be easily described by specifying the relationship between states and management operations of N. More precisely, to describe the order with which the operations of N can be executed, we introduce a transition relation τ specifying whether an operation o can be executed in a state s, and which state is reached by executing o in s.

The second kind of dependencies can be described by associating transitions and states with (possibly empty) sets of requirements to indicate that the corresponding capabilities are assumed to be provided. More precisely, the requirements associated with a transition t specify which are the capabilities that must be offered to allow the execution of t. The requirements associated with a state of a NodeType N specify which are the capabilities that must (continue to) be offered by other nodes in order for N to (continue to) work properly.

To complete the description, we also associate to each state s of a NodeType N the capabilities provided by N in s, and to explicitly specify which capabilities are maintained during a transition. The latter is a proper extension that generalises our initial definition of management protocols [1, 2], where we were assuming that all capabilities were maintained during a transition.

Definition (Management protocol)**.** *Let $N = \langle S_N, R_N, C_N, O_N, \mathcal{M}_N \rangle$ be a* NodeType*, where S_N, R_N, C_N, and O_N are the finite sets of its states, requirements, capabilities, and management operations. $\mathcal{M}_N = \langle \overline{s}_N, \rho_N, \gamma_N, \tau_N \rangle$ is the* management protocol *of N, where*

- *$\overline{s}_N \in S_N$ is the initial state,*
- *ρ_N is a function indicating, for each state $s \in S_N$, which conditions on requirements must hold (i.e., $\rho_N(s) \subseteq R_N$),*
- *γ_N is a function indicating which capabilities of N are concretely offered in a state $s \in S_N$ (i.e., $\gamma_N(s) \subseteq C_N$), and*
- *$\tau_N \subseteq S_N \times 2^{R_N} \times 2^{C_N} \times O_N \times S_N$ is a set of quintuples modelling the transition relation (i.e., $\langle s, H, G, o, s' \rangle \in \tau_N$ denotes that in state s, and if condition H holds, o is executable and leads to state s' — by maintaining the capabilities in G during the transition).*

[1] A more detailed and self-contained introduction to TOSCA can be found in [3].

```
01 <NodeType name="xs:NCName" ... >
02  ...
03  <RequirementDefinitions>
04    <RequirementDefinition name="xs:string" ... >
05       ...
06    </RequirementDefinition> +
07  </RequirementDefinitions> ?
08
09  <CapabilityDefinitions>
10    <CapabilityDefinition name="xs:string" ... >
11       ...
12    </CapabilityDefinition> +
13  </CapabilityDefinitions>
14
15  <InstanceStates>
16    <InstanceState state="xs:anyURI">
17      <ReliesOn>
18        <Requirement name="xs:string"/> +
19      </ReliesOn> ?
20      <Offers>
21        <Capability name="xs:string"/> +
22      </Offers> ?
23    </InstanceState> +
24  </InstanceStates> ?
25
26  <Interfaces>
27    <Interface name="xs:NCName|xs:anyURI">
28      <Operation name="xs:NCName"> ...  </Operation> +
29    </Interface> +
30  </Interfaces> ?
31
32  <ManagementProtocol>
33    <InitialState state="xs:anyURI"/>
34    <Transitions>
35      <Transition sourceState="xs:anyURI" targetState="xs:anyURI"
36          operationName="xs:NCName" interfaceName="xs:NCName|xs:anyURI">
37        <ReliesOn>
38          <Requirement name="xs:string"/> +
39        </ReliesOn> ?
40        <Preserves>
41          <Capability name="xs:string"/> +
42        </Preserves> ?
43      </Transition> +
44    </Transitions> ?
45  </ManagementProtocol> ?
46 </NodeType>
```

Fig. 2: Extended XML description of a `NodeType`.

Syntactically, to represent $\mathcal{M}_N$ we slightly extend the syntax for describing a TOSCA `NodeType` (Fig. 2[2]). First, we enrich the description of an `InstanceState` by introducing the nested elements `ReliesOn` and `Offers` (lines 17-22), which implement the functions ρ_N and γ_N of $\mathcal{M}_N$. More precisely, `ReliesOn` implements ρ_N by enabling the association between each instance state and the set of assumed requirements, while `Offers` implements γ_N by indicating the capabilities concretely offered in a state.

We also introduce the element `ManagementProtocol` (lines 32-42), that permits specifying the `InitialState` $\overline{s}_N$ of the protocol (line 33), as well as the `Transitions` characterizing the transition relation τ_N (lines 34-44). To effectively implement τ_N, `Transitions` permits describing the source and target state of each `Transition` $t \in \tau_N$ (line 35), the management operation that corresponds to t (line 36), the condition on requirements that must hold to fire t (lines 37-39), and the capabilities that are preserved during t (lines 40-42).

[2] In the figure, we maintain the multiplicity notation introduced in the TOSCA specification [4] (i.e., "`?`" means that an element can appear 0 or 1 times, while "`*`" and "`+`" mean that it appears at least 0 or 1 times, respectively).

Example. Consider for instance the *Server* `NodeType` in Fig. 3. It is easy to see that R_{Server} contains only one requirement (i.e., ServerContainer), that C_{Server} contains only one capability (i.e., WebAppRuntime), and that the management operations in O_{Server} are Setup, Uninstall, Run, Stop, and Configure. Suppose also that the states in S_{Server} are Unavailable, Stopped, and Working.

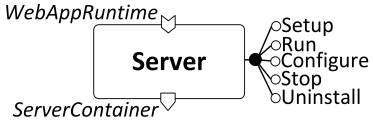

Fig. 3: Example of `NodeType`.

A possible management protocol for *Server* is shown in Fig. 4 (and the corresponding XML code is reported in Fig. 5). The initial state is Unavailable, and is not associated with any requirement or capability. Stopped is also not associated with any requirement or capability, while Working specifies that the capability corresponding to the ServerContainer requirement must be provided in order for *Server* to (continue to) work properly. State Working also specifies that Server provides the WebAppRuntime capability when in such state.

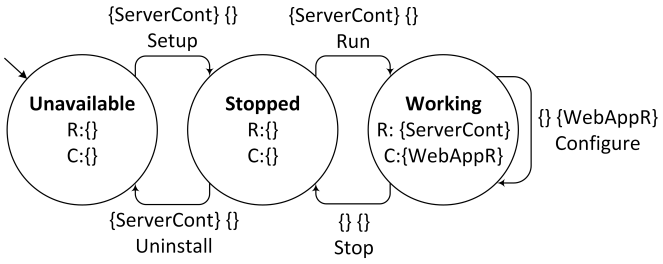

Fig. 4: Example of management protocol.

All transitions (but those involving operations Stop and Configure) bind their executability to the availability of the capability that satisfies the ServerContainer requirement. Furthermore, the only transition preserving the WebAppRuntime capability is that involving the Configure operation).

4 Concluding remarks

In this paper we have proposed an extension of TOSCA to model the behaviour of management operations and their relations with states, requirements, and capabilities. More precisely, we have properly extended the formal model[3] we proposed in [1, 2], and we have shown how to enrich the TOSCA language to permit representing such model.

[3] A detailed discussion about related and future work is out of the scope of this report. It can be found in both [1] and [2].

```
00 <NodeType name="Server" ... >
01  ...
02  <RequirementDefinitions>
03    <RequirementDefinition name="ServerContainer" ... />
04  </RequirementDefinitions>
05  <CapabilityDefinitions>
06    <CapabilityDefinition name="WebAppRuntime" ... />
07  </CapabilityDefinitions>
08  <InstanceStates>
09    <InstanceState state="Unavailable"/>
10    <InstanceState state="Stopped"/>
11    <InstanceState state="Working">
12      <ReliesOn> <Requirement name="ServerContainer"/> </ReliesOn>
13      <Offers> <Capability name="WebAppRuntime"/> </Offers>
14    </InstanceState>
15  </InstanceStates>
16  <Interfaces>
17    <Interface name="Lifecycle">
18      <Operation name="Configure"> ...  </Operation>
19      <Operation name="Run"> ...  </Operation>
20      <Operation name="Setup"> ...  </Operation>
21      <Operation name="Stop"> ...  </Operation>
22      <Operation name="Uninstall"> ...  </Operation>
23    </Interface>
24  </Interfaces>
25  <ManagementProtocol>
26    <InitialState state="Unavailable"/>
27    <Transitions>
28      <Transition sourceState="Unavailable" targetState="Stopped"
29          operationName="Setup" interfaceName="Lifecycle">
30        <ReliesOn>
31          <Requirement name="ServerContainer"/>
32        </ReliesOn>
33      </Transition>
34      <Transition sourceState="Stopped" targetState="Working"
35          operationName="Run" interfaceName="Lifecycle">
36        <ReliesOn>
37          <Requirement name="ServerContainer"/>
38        </ReliesOn>
39      </Transition>
40      <Transition sourceState="Working" targetState="Working"
41          operationName="Configure" interfaceName="Lifecycle">
42        <Preserves>
43          <Capability name="WebAppRuntime"/>
44        </Preserves>
45      </Transition>
46      <Transition sourceState="Working" targetState="Stopped"
47          operationName="Stop" interfaceName="Lifecycle"/>
48      <Transition sourceState="Stopped" targetState="Unavailable"
49          operationName="Uninstall" interfaceName="Lifecycle">
50        <ReliesOn>
51          <Requirement name="ServerContainer"/>
52        </ReliesOn>
54      </Transition>
55    </Transitions>
56  </ManagementProtocol>
57 </NodeType>
```

Fig. 5: XML fragment the management protocol in Fig. 4.

References

[1] Antonio Brogi, Andrea Canciani, and Jacopo Soldani. Modelling and analysing cloud application management. In *Proceedings of the 4th European Conference on Service-Oriented and Cloud Computing (ESOCC 2015)*, LNCS. Springer, 2015. *In press.*

[2] Antonio Brogi, Andrea Canciani, Jacopo Soldani, and PengWei Wang. Modelling the behaviour of management operations in cloud-based applications. In Daniel Moldt, editor, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15)*, volume 1372 of *CEUR Workshop Proceedings*, pages 191–205. CEUR-WS.org, 2015. *In press*, `http://www.di.unipi.it/~soldani/web/papers/2015/PNSE15.pdf`.

[3] Antonio Brogi, Jacopo Soldani, and PengWei Wang. TOSCA in a Nutshell: Promises and Perspectives. In Massimo Villari, Wolf Zimmermann, and Kung-Kiu Lau, editors, *Service-Oriented and Cloud Computing*, volume 8745 of *LNCS*, pages 171–186. Springer, 2014.

[4] OASIS. Topology and Orchestration Specification for Cloud Applications. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf`, 2013.

[5] OASIS. TOSCA Simple Profile in YAML. `http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf`, 2014.

Appendix

In this appendix, we show a possible extension of the syntax of `node_type`s (in the current version of YAML Simple Profile of TOSCA [5]) that permit describing their management protocols.

As shown in Fig. 6, we include the entries `instance_states` and `management_protocol`. As for the XML version, the instance state description allows to specify the `name` of a state, as well as the requirements it `relies_on` and the capabilities it `offers` (according to ρ and γ — lines 10-16). On the other hand, the entry `management_protocol` (lines 17-24) allows to specify the `initial_state` of the protocol (line 18) and the `transitions` characterizing its transition relation τ (lines 20-28).

```
01 <node_type_name>:
02   ...
03   requirements:
04     <requirement_definitions>
05   capabilities:
06     <capability_definitions>
07   interfaces:
08     <interface_definitions>
09   ...
10   instance_states:
11     instance_state:
12       name: <name>
13         relies_on:
14           <list_of_requirements>
15         offers:
16           <list_of_capabilities>
17    management_protocol:
18      initial state: <state_name>
19      transitions:
20        transition:
21          source_state: <state_name>
22          target_state: <state_name>
23          operation_name: <operation_name>
24          interface_name: <interface_name>
25          relies_on:
26            <list_of_requirements>
27          preserves:
28            <list_of_capabilities>
```

Fig. 6: Extended YAML description of a `NodeType`.